Δημιουργία Edges και Outline Effects στην επεξεργασία Εικόνας σε ασπρόμαυρη και σε έγρωμη

|  |  |  |
| --- | --- | --- |
| Τζίτζος Παύλος  Κουμπάνης Αθανάσιος | 7/12/2023 | Καθηγητής: Γ. Συρακούλης |

Μπορείτε να δείτε το υλικό της αναφοράς [εδώ](https://github.com/PavlosTzitzos/esd-project) .

[Λίστα Αντικειμένων 2](#_Toc1809935431)

[Λίστα Πινάκων 3](#_Toc2137822619)

[Λίστα Εικόνων 3](#_Toc2031150751)

[Θεωρητικό Υπόβαθρο 4](#_Toc965282776)

[Μετατροπή YUV σε RGB 5](#_Toc1490617241)

[Μετατροπή Έγχρωμης RGB εικόνας σε Ασπρόμαυρη 5](#_Toc2094231201)

[Εφαρμογή του Γκαουσιανού Φίλτρου στην Ασπρόμαυρη εικόνα 6](#_Toc1116208500)

[Υπολογισμός της Βαθμίδας της εικόνας 6](#_Toc230267450)

[Υπολογισμός της Έντασης και της Γωνίας των pixel της εικόνας 7](#_Toc854184374)

[Χρωματισμός των pixel βάση της Έντασης και της Γωνίας 7](#_Toc1622196498)

[Κώδικας C 7](#_Toc444214032)

[read\_image 8](#_Toc1438765575)

[write\_id\_image 8](#_Toc818206393)

[id1\_to\_id2 8](#_Toc1398320302)

[grayscale\_id 8](#_Toc1283704942)

[id1\_filter{\_id2} 9](#_Toc375918668)

[id\_calc 9](#_Toc1457401017)

[min{Number} , max{Number} 9](#_Toc50616467)

[find\_min, find\_max , linear\_scaling και scale\_magnitude\_image 10](#_Toc140937599)

[scale\_rgb\_image, find\_min\_{id}, find\_max\_{id} 10](#_Toc1345396187)

[colour\_image 10](#_Toc975801925)

[Είδη Βελτιστοποίησης 11](#_Toc1816036703)

[Είδη Βελτιστοποίησης βάση του εύρους(scope) της μεθόδου 12](#_Toc446333876)

[Βαθμωτή Βελτιστοποίηση (Scalar Optimization) 13](#_Toc511540342)

[Βελτιστοποιημένος Κώδικας και Μετρήσεις 13](#_Toc1509249170)

[Μετρήσεις με το Visual Studio IDE 14](#_Toc140648572)

[Μετρήσεις με το CodeWarrior IDE 14](#_Toc1495423251)

[Μνήμη Ενσωματωμένων Συστημάτων με χρήση ARM επεξεργαστών 16](#_Toc2006284507)

[Λογισμικό για ARM επεξεργαστή 18](#_Toc839248998)

[Αρχεία scatter.txt 19](#_Toc1888233952)

[Αρχεία \*.map 19](#_Toc624604301)

[Βελτιστοποίηση σε Επίπεδο Μνήμης 19](#_Toc1484791600)

[Αναφορές 29](#_Toc1616484576)

# Λίστα Αντικειμένων

## Λίστα Πινάκων

Πίνακας 1 - Χρώματα των pixel ανάλογα την γωνία

Πίνακας 2 - Απαιτήσεις Μνήμης Αρχικού Προγράμματος

Πίνακας 3 - Απαιτήσεις Μνήμης Βελτιστοποιημένου Προγράμματος

Πίνακας 4 - Σύνολο Εντολών Αρχικού και Βελτιστοποιημένου Προγράμματος

Πίνακας 5 - Απαιτήσεις μνήμης του βελτιστοποιημένου προγράμματος με μνήμη έκδοση v1

Πίνακας 6 - Απαιτήσεις μνήμης του βελτιστοποιημένου προγράμματος με μνήμη έκδοση v2

Πίνακας 7 - Απαιτήσεις μνήμης του βελτιστοποιημένου προγράμματος με μνήμη έκδοση v3

Πίνακας 8 - Απαιτήσεις μνήμης του βελτιστοποιημένου προγράμματος με μνήμη έκδοση v4

Πίνακας 9 - Απαιτήσεις μνήμης του βελτιστοποιημένου προγράμματος με μνήμη έκδοση v2

Πίνακας 10 - Μετρήσεις της μνήμης του βελτιστοποιημένου κώδικα v1

Πίνακας 11 - Μετρήσεις της μνήμης του βελτιστοποιημένου κώδικα v2

Πίνακας 12 - Μετρήσεις της μνήμης του βελτιστοποιημένου κώδικα v3

Πίνακας 13 - Εντολές και Κύκλοι του Επεξεργαστή μαζί με τις 3 διαφορετικές δομές μνήμης

Πίνακας 14 - Μετρήσεις της μνήμης του βελτιστοποιημένου κώδικα v3 με τις διαφορετικές δομές μνήμης

Πίνακας 15 - Εντολές και Κύκλοι των 4ων διαφορετικών δομών μνήμης

Πίνακας 16 - Μετρήσεις της μνήμης του βελτιστοποιημένου κώδικα v3 μαζί με την βέλτιστη δομή μνήμης

Πίνακας 17 - Χρόνοι Εκτέλεσης

## Λίστα Εικόνων

Εικόνα 1 - Δομή της Μνήμης που προτείνουμε

Εικόνα 2 - Διάγραμμα πλήθους πινάκων σε κάθε βήμα του αρχικού προγράμματος

Εικόνα 3 - Διάγραμμα πλήθους πινάκων σε κάθε βήμα του βελτιστοποιημένου προγράμματος

Εικόνα 4 - Διάγραμμα πλήθους πινάκων σε κάθε βήμα του βελτιστοποιημένου προγράμματος με μνήμη

Εικόνα 5 - Δομή της Μνήμης με on-chip και off-chip μνήμες

Εικόνα 6 - Δομή της Μνήμης on-chip ROM και off-chip SRAM μνήμες

# Θεωρητικό Υπόβαθρο

## Μετατροπή YUV σε RGB

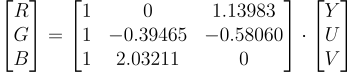
Για να μετατρέψουμε την εικόνα σε γκρι πρέπει να μετατρεψουμε το χρωματικό χώρο YUV σε RGB. Τα YUV μοντελοποιούν την εικονα χρησιμοποιώντας 3 διαφορετικα στοιχεία :

Y (luminance) για φωτεινότητα, U για το μπλε χρώμα και V για το κόκκινο.

Σε αντίθεση με το YUV μοντέλο , το RGB μοντέλο χρησιμοποιεί 3 στοιχεία για τα χρώματα:

R (red) για το κόκκινο , G (green) για το πράσινο και B (blue) για το μπλέ.

Για τη μετατροπή από YUV σε RGB χρησιμοποιούμε τις εξισώσεις :



Στο σημείο αυτό πρέπει να έχουμε υπόψιν μας ότι τα δύο αυτά μοντέλα έχουν διαφορετικό εύρος στις τιμές των στοιχείων τους. Το YUV παίρνει τις εξής τιμές :

ή

Ενώ το RGB κυμαίνεται μεταξύ των : .

Οπότε μετά από κάθε μετατροπή είναι αναγκαίο να κάνουμε κανονικοποίηση.

## Μετατροπή Έγχρωμης RGB εικόνας σε Ασπρόμαυρη

Επειδή θέλουμε να εντοπίσουμε τις ακμές στην εικόνα μας πριν εφαρμόσουμε το γκαουσιανό φίλτρο μετατρέπουμε την εικόνα από έγχρωμη RGB σε ασπρόμαυρη. Υπάρχουν 3 τρόποι με τους οποίους μπορούμε να μετατρέψουμε μια εικόνα σε ασπρόμαυρη.

Μεθοδος Φωτός



Μέθοδος Αριθμητικού Μέσου



Μέθοδος Φωτεινότητας



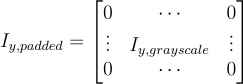
Η τελευταία είναι η πιο αποτελεσματική επειδή το μάτι αντιδρά διαφορετικά σε κάθε χρώμα οπότε βάζοντας βάρη σε κάθε τιμή RGB πετυχαίνουμε να προσαρμόσουμε την γκρι εικόνα στο πως την αντιλαμβανόμαστε .

Το εύρος τιμών των ασπρόμαυρων εικόνων κυμαίνεται από 0 έως 255.

## Εφαρμογή του Γκαουσιανού Φίλτρου στην Ασπρόμαυρη εικόνα

Επειδή ο σκοπός της εργασίας είναι να αναγνωρίσουμε τις ακμές στην εικόνα και να τις χρωματίσουμε σε επόμενο βήμα, πρέπει να εφαρμόσουμε το γκαουσιανό φίλτρο στην εικόνα. Αυτό γίνεται με συνέλιξη. Επειδή όμως εφαρμόζουμε την συνέλιξη με μάσκα 3x3 στην πρώτη γραμμή θα χρειαστούμε επιπλέον μία γραμμή. Αλλά και μία στήλη στα αριστερά της εικόνας. Στα δεξία της εικόνας αλλά και στο κάτω μέρος της εικόνας θα χρειαστούμε επίσης μια επιπλέον στήλη και γραμμή αντίστοιχα. Για αυτό τον λόγο δημιουργούμε μια νέα εικόνα .

Η επαυξημένη εικόνα είναι :



Η συνέλιξη του φίλτρου με την εικόνα είναι :

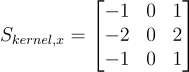


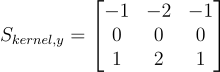
## Υπολογισμός της Βαθμίδας της εικόνας

Εφόσον έχουμε εφαρμόσει το γκαουσιανό φίλτρο υπολογίζουμε την βαθμίδα της φιλτραρισμένης εικόνας που αντιπροσωπεύει τον ρυθμό αλλαγής της έντασης της φωτινότητας του κάθε pixel. Η βαθμίδα της εικόνας ορίζεται ως:

 , όπου  και .

Για να υπολογίσουμε τις παραγώγους χρησιμποιήσαμε φίλτρα sobel και τα εφαρμόσαμε στην φιλτραρισμένη εικόνα με συνέλιξη. Οι μάσκες των φίλτρων sobel είναι :





## Υπολογισμός της Έντασης και της Γωνίας των pixel της εικόνας

Έχοντας την βαθμίδα υπολογισμένη μπορούμε να υπολογίσουμε της ένταση και την γωνία του κάθε pixel. Η ένταση αντιπροσωπεύει την ένταση του χρώματος γκρι / λευκό και είναι το μέτρο της βαθμίδας ενώ η γωνία δείχνει την κλίση της κατεύθυνσης της ένατσης πάνω σε ένα x,y επίπεδο , το επίπεδο της εικόνας. Οι τύποι για τον υπολογισμό τους είναι αντίστοιχα:





Η ένταση έχει εύρος τιμών μεταξύ 0 και 255.

Η γωνία έχει εύρος τιμών μεταξύ 0 έως 180 μοίρες.

## Χρωματισμός των pixel βάση της Έντασης και της Γωνίας

Επειδή σε μια εικόνα μια ακμή μπορεί να δείχνει προς οποιαδήποτε κατεύθυνση χρησιμοποιούμε διαφορετικά χρώματα ανάλογα την γωνία που έχουμε υπολογίσει για εκείνο το pixel και το χρωματίζουμε . Έτσι όλα τα pixel που ανήκουν επάνω στην ακμή θα έχουν ίδιο χρώμα και η ένταση του χρώματος αυτού εξαρτάται από την ένταση (μέτρο) της κλίσης που υπολογίσαμε προηγουμένως. Ορίζουμε τις εξής περιοχές:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Στρογγυλοποίηση () | Χρώμα ομάδας | Ακμές |
| 0 | 22.5 | 0 | κίτρινο | Κατακόρυφες |
| 22.5 | 67.5 | 45 | πράσινο | Διαγώνιες προς δεξιά |
| 67.5 | 112.5 | 90 | μπλέ | Οριζόντιες |
| 112.5 | 157.5 | 135 | κόκκινο | Διαγώνιες προς αριστερά |
| 157.5 | 180 | 0 | κίτρινο | Κατακόρυφες |

*Πίνακας 1 - Χρώματα των pixel ανάλογα την γωνία*

# Κώδικας C

Ο κώδικας υλοποιήθηκε αρχικά χρησιμοποιώντας το Visual Studio για να μπορέσουμε να κάνουμε debugging εύκολα και το github για να μπορέσουμε να διατηρήσουμε διαφορετικές εκδόσεις λόγω βελτιστοποιήσεων αλλά και για να μπορούμε να δουλέψουμε από διαφορετικές συσκευές στην ίδια εργασία. Η εικόνα με την οποία δοκιμάσαμε τον κώδικα μας είχε διαστάσεις 496x376 και ήταν της μορφής YUV 420.

## *read\_image*

1. Το πρώτο βήμα στην εκτέλεση του προγράμματος είναι να φορτώσουμε την εικόνα από την μνήμη. Η συνάρτηση αυτή κάνει αυτή την δουλειά , με παράμετρο filename που έχουμε δηλώσει στην αρχή του αρχείου.

## *write\_id\_image*

1. Προκειμένου να ελέγξουμε μετά από κάθε αλγόριθμο το αποτέλεσμα γράψαμε συναρτήσεις για αυτόν τον σκοπό. Τα ονόματα αυτών των συναρτήσεων έχουν την παρακάτω δομή:
2. void write\_*id*\_image() ,
3. όπου *id* είναι το αναγνωριστικό της μεθόδου που ελέγχουμε.

## *id1\_to\_id2*

1. Χρειάστηκε σε πολλά σημεία να μετατρέψουμε την εικόνα από μια μορφή σε μια άλλη. Οι συναρτήσεις αυτής της δομής εκτελούν αυτές τις μετατροπές:
2. void *id1*\_to\_*id2*() ,
3. όπου *id1* είναι το χρωματικό μοντέλο που διαθέτουμε και *id2* είναι το χρωματικό μοντέλο στο οποίο θέλουμε να μετατρέψουμε την εικόνα μας.

## *grayscale\_id*

1. Οι συναρτήσεις αυτές χρησιμοποιούνται από την void rgb\_to\_grayscale(int sel) που ανοίκει στην παραπάνω κατηγορία συναρτήσεων. Η ιδιαιτερότητα αυτής της συνάρτησης είναι δίνει την δυνατότητα να διαλέξουμε ποια μέθοδο επιθυμούμε να χρησιμοποιήσουμε κατά την μετατροπή μιας έγχρωμης rgb εικόνας σε ασπρόμαυρη. Οι συναρτήσεις αυτές έχουν την παρακάτω δομή :
2. void grayscale\_*id*() ,
3. όπου *id1* είναι το χρωματικό μοντέλο που διαθέτουμε και *id2* είναι το χρωματικό μοντέλο στο οποίο θέλουμε να μετατρέψουμε την εικόνα μας.

## *id1\_filter{\_id2}*

1. Όπως δηλώνει το όνομα filter οι συναρτήσεις αυτές υλοποιούν τις συνελίξεις των εικόνων με τις μάσκες (kernels). Οι συναρτήσεις αυτές έχουν την παρακάτω δομή :
2. void *id1*\_filter{\_*id2*}() ,
3. όπου *id1* είναι ο τύπος του φίλτρου που εφαρμόζουμε πχ γκαουσιανό (gaussian) και το *id2* είναι μια παράμετρος που χρησιμοποιείται από τα φίλτρα sobel για να δείξει αν μιλάμε για την παράγωγο ως προς x ή ως προς y.

## *id\_calc*

1. Για τους υπολογισμούς που χρησιμοποιούν την βαθμίδα οι συναρτήσεις έχουν την μορφή αυτή :
2. void *id*\_calc() ,
3. όπου *id* είναι ο τύπος της μαθηματικής ιδιότητας της βαθμίδας πχ angle
   1. Την ίδια δομή έχει και η συνάρτηση που υπολογίζει την βαθμίδα :
   3. void gradient\_calc()
   5. Μια ματιά στο εσωτερικό αυτής της συνάρτησης :
   6. Η συνάρτηση αυτή καλεί τις συναρτήσεις φίλτρων sobel ( sobel\_filter\_x και sobel\_filter\_y )

## *min{Number} , max{Number}*

1. Επειδή σε διάφορα σημεία χρειάστηκε να υπλογισθεί το ελάχιστο ή το μέγιστο στοιχείο μεταξύ 3ων στοιχείων υλοποιήθηκαν οι συναρτήσεις αυτές που δίνουν το αποτέλεσμα αυτό :
2. void min{*Number*}() ,
3. όπου Number είναι ο αριθμός των στοιχείων που συγκρίνει πχ 2 για να βρει τον ελάχιστο μεταξύ τους.
4. Ομοίως για την :
6. void max{*Number*}()
   1. Μια ματιά στο εσωτερικό των συνάρτησεων min3 , max3 :
   2. Οι συναρτήσεις αυτές καλούν τις συναρτήσεις min2 , max2 αντίστοιχα.
   4. Μια ματιά στο εσωτερικό των συνάρτησεων min2 , max2 :
   5. Οι συναρτήσεις αυτές συγκρίνουν τα 2 στοιχεία που τους έχουν δωθεί και επιστρέφουν το μικρότερο ή το μεγαλύτερο από τα 2. Σε περίπτωση που είναι ίσα επιστρέφουν -1. Αυτη την περίπτωση την διαχειρίζονται οι συναρτήσεις που τις κάλεσαν.

## *find\_min, find\_max , linear\_scaling και scale\_magnitude\_image*

1. Προκειμένου να κάνουμε κανονικοποίηση στις τιμές των pixel της εικόνας εφαρμόζουμε μια απλή ευθεία γραμμή :
2. 
3. όπου *id1* είναι ο τύπος του φίλτρου που
4. Πριν όμως εφαρμόσουμε την ευθεία γραμμή πρέπει να βρούμε το ελάχιστο και το μέγιστο των τιμών των pixel για αυτό χρησιμοποιούμε τις συναρτήσεις find\_min() και find\_max().

## *scale\_rgb\_image, find\_min\_{id}, find\_max\_{id}*

1. Όταν μετατρέπουμε την εικόνα από YUV σε RGB δεν αρκεί να βρούμε τις τιμές των χρωμάτων RGB μόνο. Πρέπει να προσέξουμε το εύρος των τιμών τους να είναι σύμφωνο με την θεωρία όπως έχει αναφερθεί παραπάνω. Για αυτόν τον λόγο η συνάρτηση scale\_rgb\_image εφαρμόζει κανονικοποίηση πάνω σε κάθε τιμή των RGB πινάκων. Οι συναρτήσεις find\_min\_{id} και find\_max\_{id} βρίσκουν τη μικρότερη και τη μεγαλύτερη τιμή του πίνακα με αναγνωριστικό το id. Το id μπορεί να είναι r , g ή b ανάλογα σε ποιο frame εφαρμόζουμε την αναζήτηση. Η αναζήτηση είναι η πιο απλή αλγοριθμικά συγκρίνει όλα τα στοιχεία μεταξύ τους. Φυσικά υπάρχει περιθώριο αλλαγής.

## *colour\_image*

1. Το τελευταίο βήμα του απρογράμματος είναι να βάψει την εικόνα σύμφωνα με την γωνία και την ένταση(μέτρο) της βαθμίδας που έχει κάθε pixel.

# Είδη Βελτιστοποίησης

Στην προσπάθεια να γράψουμε το παραπάνω πρόγραμμα χρησιμοποιώντας εργαλεία που μετράνε την κατανάλωση μνήμης , επεξεργαστή , χρόνου , πράξεων στον επεξεργαστή και άλλων χρήσιμων πόρων όπως έχει το Visual Studio αλλά και το Armulator παρατηρήσαμε ότι υπήρχαν υψηλές καταναλώσεις μνήμης και άλλων πόρων. Θα θέλαμε να κάνουμε τον κώδικά μας πιο ποιοτικό και για να το πετύχουμε αυτό έχουμε στη διάθεση μας διάφορες τεχνικές βελτιστοποίησης του κώδικά μας ανάλογα την μετρική που θέλουμε να αυξήσουμε ή να μειώσουμε.

Η ποιότητα κώδικα μπορεί να σημαίνει πολλά πράγματα και μπορεί να μετρηθεί με πολλούς τρόπους και σε διαφορετικά στάδια. Αρκεί να αναφέρουμε μερικά παραδείγματα:

1. Ταχύτητα Εκτέλεσης (Runtime Speed)
2. Μέγεθος Κώδικα
3. Ενέργεια που καταναλώνει ο επεξεργαστής κατά της εκτέλεση

Παρακάτω παρουσιάζονται ορισμένες τεχνικές Βελτιστοποίησης συνοπτικά.

## Είδη Βελτιστοποίησης βάση του εύρους(scope) της μεθόδου

1. Τοπική Βελτιστοποίηση (Local Optimization) , που βασίζεται στην μέθοδο Απαρίθμησης Τοπικών Τιμών (Local Value Numbering - LVN) και αφορά μικρά κομμάτια κώδικα που μπορούν να εκτελεστούν σειριακά. Στην μέθοδο αυτή βασίζονται οι παρακάτω :
   1. Αναδήπλωση Σταθερών (Constant folding)
   2. Αλγεβρική Απλοποίηση (Algebraic simplification)
2. \* Οι αλγόριθμοι αυτοί είναι εκτός του εύρους της συγκεκριμένης εργασίας αλλά προστέθηκαν για αναφορά μόνο.
3. Βελτιστοποίηση Περιοχής (Regional or Loop-Level Optimization). Αφορούν περιοχές στο πρόγραμμα που περιλαμβάνουν βρόχους και δομές ελέγχου. Μερικές μέθοδοι είναι :
   1. Απαρίθμηση Υπερ-τοπικών Τιμών (Superlocal Value Numbering), είναι μια επέκταση της LVN
   2. Ξεδίπλωμα Βρόχου (Loop unrolling)
4. Βελτιστοποίηση Προγράμματος (Global Optimization). Λόγω του μεγάλου εύρους (scope) και ότι περιλαμβάνουν περιοχές με βρόχους και δομές ελέγχου οι βελτιστοποιήσεις αυτές εφαρμόζονται κατά το στάδιο της ανάλυσης πριν ξαναγράψουν τον κώδικα.
   1. Εντοπισμός Μη αρχικοποιημένων μεταβλητών με ζωντανές δομές (Finding Uninitialized Variables with Live Sets)
   2. Γενικευμένη Τοποθέτηση Κώδικα (Global Code Placement)
5. Βελτιστοποίηση Διαδικασιών (Interprocedural Optimization)
   1. Αντικατάσταση Inline μεθόδων (Inline Substitution)
   2. Τοποθέτηση Διεργασιών (Procedure Placement)

## Βαθμωτή Βελτιστοποίηση (Scalar Optimization)

1. Η βαθμωτή βελτιστοποίηση αφορά την βελτιστοποίηση του κώδικα που έχει αναλάβει ένα νήμα (thread) ελέγχου. Βασίζεται στις παραπάνω μεθόδους βελτιστοποίησης αλλά και στην Στατική Ανάλυση[ref]. Οι τεχνικές που αφορούν την μέθοδο αυτή είναι:
2. Καθαρισμός Άχριστου και Απροσπέλαστου Κώδικα
3. Μετακίνηση Κώδικα (Lazy Code Motion)
4. Εξιδίκευση Υπολογισμού
5. Απαλοιφή Περιττών Υπολογισμών
6. Ενεργοποίηση άλλων Μετασχηματισμών[ρεφ] , όπως είναι ο μετασχηματισμός loop unswitching που βγάζει έξω από έναν βρόχο τις δομές ελέγχου που δεν μεταβάλλονται (loop-invariant control-flow)
7. Στο σημείο θα αναφέρουμε τις μεθόδους βελτιστοποίησης που μας ενδιαφέρουν περισσότερο σε αυτή τη φάση της εργασίας:
8. Loop Unrolling
9. Loop Fusion - Loop Fission
10. Code Motion
11. Strength Reduction
12. Dead Code Elimination
13. Common Subexpression Elimination (CSE)
14. Function Inlining – Inline Expansion
15. Constant Folding - Copy Propagation

# Βελτιστοποιημένος Κώδικας και Μετρήσεις

Για να πετύχουμε υψηλή ποιότητα στον κώδικα αρχικά θέλαμε να είναι όλες οι συναρτήσεις inline σαν πρώτο βήμα. Όμως αυτό δεν είναι εφικτό σε όλες για αυτό και τοποθετήσαμε μόνο τα βασικά blocks μέσα στην main. Το επόμενο βήμα είναι να πετύχουμε βελτιστοποίηση στην κατανάλωση του επεξεργαστή για και ειδικά στα πιο κρίσιμα σημεία. Το visual studio βοηθάει με τον CPU profiler σαν πρώτη εικόνα. Επίσης το ίδιο IDE δίνει την δυνατότητα να βλέπουμε την Process Memory με τα private bytes. Έτσι μπορούμε δοκιμάζοντας τον κώδικα να μειώσουμε την κατανάλωση μνήμης που χρειάζεται.

## Μετρήσεις με το Visual Studio IDE

Σύμφωνα με το Visual Studio το μη-βελτιστοποιημένο πρόγραμμα χρρειάζεται 26 MB και χρειάζεται 2.4 s για να εκτελεστεί.Το βελτιστοποιημένο πρόγραμμα χρειάζεται 24 MB και εκτελείται σε 2.5553 s όταν έχουμε σβήσει όλα τα σχόλια , έχουμε κρατήσει μόνο τις συναρτήσεις που είναι απολύτως απαραίτητες για την παραγωγή του σωστού αποτελέσματος και εφόσον τις ενσωματώσαμε όλες στην main.

Αναπτύσσοντας την συνάρτηση padder() σε 3 διαφορετικούς βρόχους έριξε την επεξεργαστική ισχύ από 25% που ήταν προηγουμένως και για τον μη-βελτιστοποιημένο και για τον βελτιστοποιημένο κώδικα σε 22% κατά την εκτέλεση της συγκεκριμένης συνάρτησης.

Έπειτα εφαρμόζουμε loop unrolling στην συνάρτηση που κάνει κανονικοποίηση το κόκκινο frame αφού του φορτώσουμε δεδομένα από την εικόνα YUV.

Η υψηλή κατανάλωση της μνήμης επιχειρήσαμε να την μειώσουμε δεσμεύοντας όλο και λιγότερη μνήμη στις global μεταβλητές.

Από 26ΜΒ σε 24ΜΒ βάζοντας όλες τις απαραίτητες συναρτήσεις μέσα στην main.

Από 24MB σε 21ΜΒ μειώνοντας κατά 3 τους πίνακες (σβήσαμε τους frame\_grayscale\_r , g , b) και κρατήσαμε μόνο τους πίνακες που αρχηκά λεγόντουσαν frame\_y , frame\_u , frame\_v και frame\_r , frame\_g , frame\_b και τους μετονομάσαμε σε frame\_1\_a , frame\_1\_b frame\_1c και frame\_2\_a frame\_2\_b frame\_2\_c

Από 21 ΜΒ σε 17ΜΒ σβήνοντας τους πίνακες coloured\_image που χρησιμοποιύνται στο βήαμ 7 και τους αντικαταστήσαμε με τους παραπάνω frame\_1\_{a,b,c} και frame\_2\_{a,b,c}.

## Μετρήσεις με το CodeWarrior IDE

Χρειάστηκε να γίνουν κάποιες αλλαγές στον κώδικα , προκειμένου να μπορέσει να εκτελεστεί σωστά από το CodeWarrior IDE. Αυτές είναι η περιοχή pragma arm section και οι διορθώσεις των pow(), arctan() διότι είχαν unhandled overflow. Επίσης χρειάστηκαν τα αρχεία scatter.txt, memory.map και stack. Το πρώτο δείχνει τον χώρο αποθήκευσης (δομή μνήμης στο λογισμικό) , το δεύτερο την δομή της μνήμης στο υλικό και το τρίτο δηλώνει την δομή δεδομένων την οποία έχει η μνήμη.

Οι μέθοδοι βελτιστοποίησης που χρησιμοποιήσαμε ήταν διαδοχικά :

1. Inline Substitution – Code Motion. Αντί για να βάλουμε inline μπροστά από τις συναρτήσεις τις βάλαμε μέσα στην main γιατί παρατηρήσαμε ότι βελτίωνε τις μετρικές του προγράμματος.
2. Loop unrolling. Στους περισσότερους βρόχους εφαρμόσαμε την τεχνική αυτή.
3. Loop fussion. Στον βρόχο που δημιουργεί την επαυξημένη εικόνα εφαρμόσαμε την τεχνική αυτή. Σπάσαμε την δομή for-for-if-else σε for-for , for, for.
4. Loop fission. Όταν εκτελούμε μετατροπές από ένα χρωματικό χώρο σε άλλο κρατήσαμε τους βρόχους ενωμένους.
5. Dead Code Elimination. Υπήρχαν κάποια μονοπάτια στον κώδικα που δεν εκτελούνταν ποτέ. Τα απαλειψαμε.
6. Strength Reduction. Πολλές αλγεβρικές πράξεις που υπήρχαν στον κώδικα τις κόψαμε σε δύο κομμάτια μειώνοντας την βαρύτητα της συνολικής πράξης.
7. Copy Propagation. Στα #define και στην μεταβλητή neighborhood\_of\_image αντικαταστήσαμε αρχικά τις τιμές πχ filename με τη πραγματική τιμή. Όμως η neighborhood\_of\_image είχε κι άλλα περιθώρια βελτιστοποίησης και ενώ την κάναμε να φορτώνει δεδομένα από τον πίνακα και να αρχικοποιείται τελικά ανοιγωντά την στον χώρο (κάθε στοιχείο φορτώνεται σε ξεχωριστή γραμμή) είχε πιο πολύ κόστος. Τελικά με τις δύο for πετύχαμε βελτιστοποίηση του κώδικα.
8. Common Subexpression Elimination. Οι τιμές των μεταβλητών slope\_id υπολογίζονται στην αχή του προγράμματος και όχι μέσα στην for-for δομή όπου και χρησιμοποιούνται.

Τα στοιχεία του αρχικού προγράμματος φαίνονται παρακάτω:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| initial | Object | 7248 | 60 | 108 | 22197156 | 24016 |
| Library | 23180 | 610 | 0 | 300 | 8216 |
| Grand | 30428 | 670 | 108 | 22197456 | 32232 |
| RO | 31098 | 30.36914063 | kB |  |  |
| RW | 22197564 | 21.16924667 | MB |  |  |
| ROM | 31206 | 30.47460938 | kB |  |  |

*Πίνακας 2 - Απαιτήσεις Μνήμης Αρχικού Προγράμματος*

Ενώ του τελικού είναι:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| optimized-1 | Object | 6240 | 168 | 0 | 0 | 10364 |
| Library | 17664 | 466 | 0 | 300 | 7240 |
| Grand | 23904 | 634 | 0 | 300 | 17604 |
| RO | 24538 | 23.96289063 | kB |  |  |
| RW | 300 | 0.29296875 | kB |  |  |
| ROM | 24538 | 23.96289063 | kB |  |  |

*Πίνακας 3 - Απαιτήσεις Μνήμης Βελτιστοποιημένου Προγράμματος*

Και οι εντολές και οι κύκλοι στον επεξεργαστή είναι:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| NO MEMORY | Instuctions | Core Cycles | S Cycles | N Cycles | I Cycles | C Cycles | Total |
| initial | 620226123 | 102298801 | 728855498 | 232897531 | 120254198 | 0 | 1082007227 |
| optimized-1 | 844442376 | 1287663527 | 974504597 | 243573763 | 167334313 | 0 | 1385412673 |

*Πίνακας 4 - Σύνολο Εντολών Αρχικού και Βελτιστοποιημένου Προγράμματος*

Παρατηρούμε ότι έχουμε αύξηση των εντολών και των κύκλων εκτέλεσης. Συνεπώς δεν μπορούμε να κάνουμε πολλά για να βελτιστοποιήσουμε τον κώδικα έτσι όπως είναι αυτή τη στιγμή. Εφαρμόζοντας όλες τις παραπάνω τεχνικές δεν λύνεται το βασικότερο πρόβλημα που είναι η επαναχρησιμοποίηση μεταβλητών μέσα στην cache. Αν είχαμε την δυνατότητα να έχουμε μόνο ένα πίνακα της εικόνας δίπλα στον επεξεργαστή, τότε το πλήθος των εντολών θα ήταν πάρα πολύ μικρότερο.

# Μνήμη Ενσωματωμένων Συστημάτων με χρήση ARM επεξεργαστών

Έχοντας πλέον αναπτύξει τον κώδικα του προγράμματος εκτελώντας τα παρακάτω βήματα:

1. Αρχικές Προδιαγραφές
2. Υλοποίηση και δοκιμή κώδικα
3. Εξήγηση κώδικα
4. Βελτιστοποίηση Κώδικα
5. Μετρήσεις για σύγκριση

θα κοιτάξουμε το επόμενο κομμάτι που είναι η μνήμη.

Στην περίπτωση μας, μας ενδιαφέρει μόνο η μνήμη μέσα στον επεξεργαστή (on-chip). Η μνήμη αυτή αποτελείται από δύο κομμάτια: την μνήμη ROM στην οποία θα έχουμε τον κώδικα του προγράμματος και τα αρχικά δεδομένα (Zero-Initialized Data) και την μνήμη RAM στην οποία θα έχουμε τις δομές heap και stack και μερικά δεδομένα που προκύπτουν από το πρόγραμμα καθώς λειτουργεί.

Η διευθυνσιοδότηση της μνήμης είναι δομημένη σειριακά:

1. ROM: 0x000000 ~ ROM\_MAX
2. RAM: ROM\_MAX ~ RAM\_MAX

Μέσα στην RAM έχουμε τις δομές heap και stack. Κάθε μία από αυτές διαθέτει μια διεύθυνση βάσης από όπου ξεκινάνε να αυξάνονται σε μέγεθος. Οι διευθύνσεις αυτές πρέπει να δηλώθούν ώστε να έχουμε ολοκληρωμένη εικόνα του μοντέλου μνήμης του συστήματος μας. Είναι σημαντικό να μην ταυτίζονται και πρέπει να προβλέψουμε να μην επικαλυφθούν οι δύο δομές καθώς μεγαλώνουν σε μέγεθος.

Στο αρχικό πρόγραμμα , το μη-βελτιστοποιημένο τοποθετούμε μνήμη και έχουμε συνολικά 256 MB:

1. ROM: 0x00 000 000 ~ 0x04 000 000 = 64 MB
2. RAM: 0x04 000 000 ~ 0x10 000 000 = 192 MB

Στην RAM θα δηλώσουμε και τις διευθύνσεις της βάσης της δομής heap και της στοίβας :

1. Βάση heap: 0x6 000 000
2. Βάση στοίβας: 0x8 000 000

Οπότε δίνουμε χώρο στην heap να αναπτυχθεί είτε προς τα πάνω (προς τις μικρότερε διευθύνσεις) είτε προς τα κάτω(μεγαλύτερες διευθύνσεις) και την stack να χρησιμοποιήσουν μέχρι 32 MB. Τα υπόλοιπα 64 MB είναι Zero-Initialized data και global μεταβλητές και είναι χώρος που αφορά τους πίνακες των frame.

Στο βελτιστοποιημένο πρόγραμμα που έχουμε από την προηγούμενη ενότητα πάμε να προσθέσουμε μνήμη. Επειδή χρειαζόμαστε περίπου 16 MB για την ROM και 48 MB (συνολικά περίπου 64 MB) για την RAM δηλώνουμε αντίστοιχα τις διευθύνσεις ROM\_MAX και RAM\_MAX ως 0χ1 000 000 και 0x4 000 000. Έτσι έχουμε:

1. ROM: 0x0 000 000 ~ 0χ1 000 000 = 16 MB
2. RAM: 0χ1 000 000 ~ 0x4 000 000 = 48 MB

Στην RAM θα δηλώσουμε και τις διευθύνσεις της βάσης της δομής heap και της stack:

1. Βάση heap: 0x1 000 000
2. Βάση στοίβας: 0x2 000 000

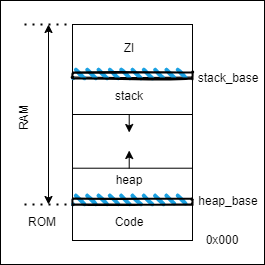
Οπότε δίνουμε χώρο στην heap και την stack να χρησιμοποιήσουν μέχρι 16 MB.

Τα υπόλοιπα 32 MB είναι Zero-Initialized data και είναι χώρος που αφορά τους πίνακες των frame.

Για τις μνήμες πέρα από το μέγεθος τους μπορούμε να καθορίσουμε και την διεπαφή τους. Μπορούμε να δηλώσουμε το πλάτος του διαύλου επικοινωνίας και τους χρόνους μη-διαδοχικής και διαδοχικής προσπέλασης. Στις παραπάνω ρυθμίσεις είχαμε :

1. ROM: δίαυλο 4bytes, μη-διαδοχική/διαδοχική προσπέλαση 1/1
2. RAM: δίαυλο 4bytes, μη-διαδοχική/διαδοχική προσπέλαση 250/50

Ένα σχεδιάγραμμα της μνήμης μπορεί να είναι το παρακάτω:



*Εικόνα 1 - Δομή της Μνήμης που προτείνουμε*

## Λογισμικό για ARM επεξεργαστή

Η διαδικασία υλοποίησης λογισμικού για επεξεργαστές της ARM είναι μια ροή που αποτελείται από τα παρακάτω βήματα:

1. Κώδικας C/C++/Java/Python (.c , .h, .cpp, .hpp, .py, .java, .jar)
2. Κώδικας Μηχανής (.o , .obj)
3. Εικόνα .axf

Ο κώδικας που γράφουμε περνάει από τον μεταγλωτιστή (compiler) και παράγει τον κώδικα μηχανής, ο οποίος με την σειρά του περάνει από τον ARM Linker και παίρνουμε το τελικό εκτελέσιμο αρχείο σε μορφή .axf . Το αρχείο .axf μπορούμε να το εξετάσουμε με τον debugger και να πάρουμε διάφορες μετρήσεις και δεδομένα για τις εντολές , τους κύκλους του επεξεργαστή και τους χρόνους τους συστήματος.

Τα αρχεία scatter.txt , mem.c χρησιμοποιούνται από τον compiler ώστε να τοποθετηθούν σωστά τα δεδομένα μέσα στα αρχεία .o (ή .obj).

Τα αρχεία memory.map χρησιμοποιούνται από τον debugger ώστε η εικόνα .axf να μπορέσει να τρέξει με τις καθορισμένες ταχύτητες, χρόνους.

## Αρχεία scatter.txt

Είναι αρχεία που δείχνουν την δομή της μνήμης. Πόσες μνήμες, τι ονόματα έχουν και το εύρος των διευθύνσεων τους. Όταν παράγεται ο κώδικας μηχανής ο compiler έχει χρησιμοποιήσει την δομή που του δώσαμε και τοποθέτησε τα δεδομένα στις αντίστοιχες θέσεις. Όταν θα παραχθεί η εικόνα .axf έχει δηλωθεί η μνήμη και μάλιστα έχουν δηλωθεί δύο μνήμες:

1. LOAD MEM: είναι η μνήμη που είναι συμπιεσμένος ο εκτελέσιμος κώδικας
2. EXECUTE MEM: είναι η μνήμη που θα τοποθετηθούν όλα τα δεδομένα όπως έχουμε δηλώσει στο αρχείο scatter.txt κατά την αποσυμπίεση.

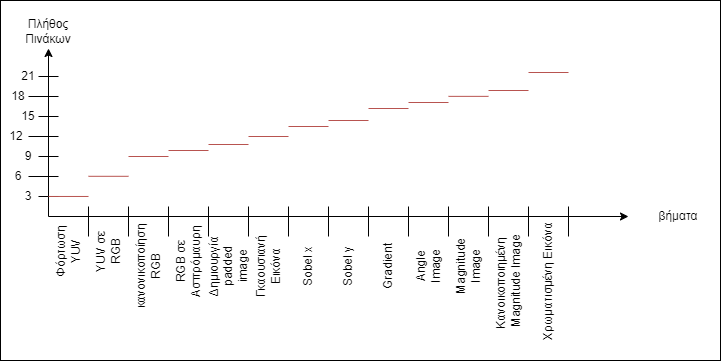
## Αρχεία \*.map

Είναι αρχεία που δείχνουν τα χαρακτηριστικά των δηλωμένων μνημών. Κατά την εκτέλεση μέσω του debugger του .axf αρχείου πρέπει να δηλώσουμε τα χαρακτηριστικά του επεξεργαστή και της μνήμης. Για τους επεξεργαστές έρχονται με το λογισμικό. Για την μνήμη μπορούμε να χρησιμοποιήσουμε την προεπιλεγμένη επιλογή (μνήμη του υπολογιστή) ή να δηλώσουμε εμείς την επιθυμιτή μνήμη μέσω των αρχείων .map . Μια γραμμή σε αυτό το αρχείο περιλμβάνει:

1. Αρχική Διεύθυνση μνήμης (να συμβαδίζει με το αρχείο scatter.txt)
2. Τελική Διεύθυνση μνήμης (να συμβαδίζει με το αρχείο scatter.txt)
3. Όνομα μνήμης (να συμβαδίζει με το αρχείο scatter.txt)
4. Εύρος Διάυλου επικοινωνίας σε bytes
5. Τύπος μνήμης ως προς την ανάγνωση και εγγραφή (R, RW)
6. Χρόνος Ανάγνωσης (ns): Μη διαδοχικής Προσπέλασης / Διαδοχικής Προσπέλασης
7. Χρόνος Εγγραφής (ns): Μη διαδοχικής Προσπέλασης / Διαδοχικής Προσπέλασης

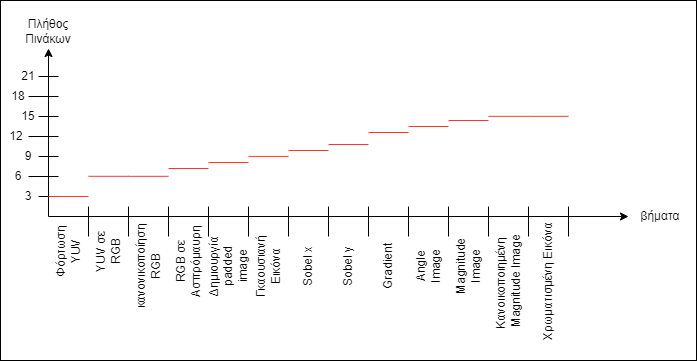
# Βελτιστοποίηση σε Επίπεδο Μνήμης

Διαπιστώσαμε ότι η ανάγκες του προγράμματος σε μνήμη είναι πολύ μεγάλες. Για να μειώσουμε αυτές τις ανάγκες θα προσπαθήσουμε να μειώσουμε το πλήθος πινάκων σε συνάρτηση με τον χρόνο. Η χρήση της μνήμης στον αρχικό κώδικα ήταν:



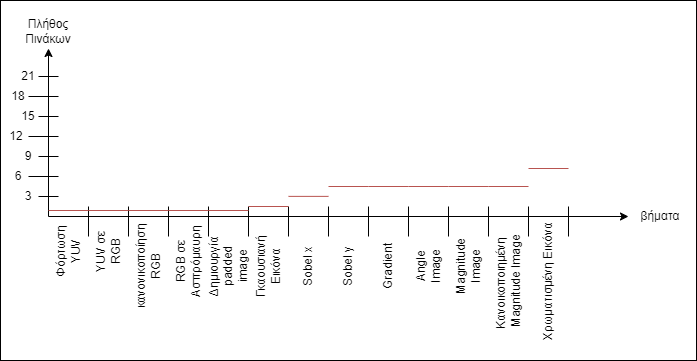
*Εικόνα 2 - Διάγραμμα πλήθους πινάκων σε κάθε βήμα του αρχικού προγράμματος*

Η χρήση 21 πινάκων χωρίς να επαναχρησιμοποιούμε κάποιους από αυτούς δεν είναι επιθυμιτό. Στην αμέσως επόμενη βελτιωμένη έκδοση έχουμε:



*Εικόνα 3 - Διάγραμμα πλήθους πινάκων σε κάθε βήμα του βελτιστοποιημένου προγράμματος*

Από τους 21 το πλήθος των πινάκων έπεσε σε 16. Πάλι όμως ήταν πάρα πολλοί για μια εφαρμογή ενσωματωμένων συστημάτων πόσο μάλλον για την επεξεργασία μιας εικόνας. Στην τελευταία έκδοση (optimized-2) έχουμε μειώσει το πλήθος των πινάκων σε 7:



*Εικόνα 4 - Διάγραμμα πλήθους πινάκων σε κάθε βήμα του βελτιστοποιημένου προγράμματος με μνήμη*

Παρακάτω εξηγούμε τα βήματα που ακολουθήσαμε και καταλήξαμε στην τελική μορφή.

Αρχικά, για να μειώσουμε τον χώρο που καταναλώνουμε από τα Zero-Initialized δεδομένα πετάμε τις περιττές αρχικοποιήσεις. Έπειτα παρατηρούμε ότι οι μάσκες των φίλτρων kernel έχουν διαφορά μεταξύ τους 90 μοίρες είναι δηλαδή περιστραμένα δεξιόστροφα. Επομένως δεν χρειάζεται να έχουμε ξεχωριστό πίνακα αρκεί να διατρέξουμε τα στοιχεία του ίδιο πίνακα κατά γραμμή ή κατά στήλη ανάλογα ποιον θα κρατήσουμε. Εμείς κρατήσαμε την μάσκα x οπότε τον ίδιο πίνακα τον διατρέχουμε κατά γραμμές όταν πρόκειται για το φίλτρο χ και κατά στήλες όταν πρόκειται για το φίλτρο y. Επίσης, αντί να φορτώνουμε τα τρία frames Υ, U, V ενώ χρειαζόμαστε μόνο την γκρί εικόνα, φορτώνουμε μόνο το Υ frame και το κανονικοποιούμε για να είμαστε μεταξύ των τιμών 0 έως 255. Τα αποτελέσματα φαίνονται παρακάτω:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| optimized-2-v1 | Object | 3936 | 112 | 0 | 0 | 9748 |
| Library | 17596 | 466 | 0 | 300 | 7164 |
| Grand | 21532 | 578 | 0 | 300 | 16912 |
| RO | 22110 | 21.59179688 | kB |  |  |
| RW | 300 | 0.29296875 | kB |  |  |
| ROM | 22110 | 21.59179688 | kB |  |  |

*Πίνακας 5 - Απαιτήσεις μνήμης του βελτιστοποιημένου προγράμματος με μνήμη έκδοση v1*

Έπειτα αντί να χρησιμοποιούμε 3 πίνακες, έναν για να φορτώνουμε τα δεδομένα, έναν για την κανονικοποίηση και έναν για την επαυξημένη εικόνα, μπορύμε να χρησιμοποιήσουμε μόνο έναν. Κάνοντας τις απαραίτητες αλλαγές έχουμε :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| optimized-2-v2 | Object | 3936 | 112 | 0 | 0 | 9764 |
| Library | 17596 | 466 | 0 | 300 | 7164 |
| Grand | 21532 | 578 | 0 | 300 | 16928 |
| RO | 22110 | 21.59179688 | kB |  |  |
| RW | 300 | 0.29296875 | kB |  |  |
| ROM | 22110 | 21.59179688 | kB |  |  |

*Πίνακας 6 - Απαιτήσεις μνήμης του βελτιστοποιημένου προγράμματος με μνήμη έκδοση v2*

Επειδή τους πίνακες frame\_padded και frame\_filtered\_y δεν τους χρειαζόμαστε εφόσον υπολογίσουμε τα φίλτρα sobel μπορούμε να τους επαναχρησιμοποιήσουμε για την γωνία και το πλάτος. Οπότε αντί να χρησιμοποιούμε ξεχωριστά array για αυτές τις διαδικασίες επαναχρησιμοποιούμε ήδη δεσμευμένες θεσης μνήμης. Κάνοντας τις απαραίτητες αλλαγές έχουμε τα παρακάτω αποτελέσματα:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| optimized-2-v3 | Object | 3200 | 112 | 0 | 0 | 9100 |
| Library | 17596 | 466 | 0 | 300 | 7164 |
| Grand | 20796 | 578 | 0 | 300 | 16264 |
| RO | 21374 | 20.87304688 | kB |  |  |
| RW | 300 | 0.29296875 | kB |  |  |
| ROM | 21374 | 20.87304688 | kB |  |  |

*Πίνακας 7 - Απαιτήσεις μνήμης του βελτιστοποιημένου προγράμματος με μνήμη έκδοση v3*

Συγκεντρωτικά φαίνονται όλα παρακάτω:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| WITH MEMORY | Instuctions | Core Cycles | S Cycles | N Cycles | I Cycles | C Cycles | Wait Cycles | Total | True Idle Cycles |
| initial | 620,226,214 | 1,022,989,014 | 728,855,589 | 232,897,632 | 120,254,219 | 0 | 1,240,908,894 | 2,322,916,334 | 35,284,909 |
| optimized-1 | 844,024,822 | 1,286,099,875 | 973,524,658 | 243,016,796 | 168,038,328 | 0 | 1,291,521,998 | 2,676,101,780 | 51,433,781 |
| optimized-2-v1 | 560,637,052 | 857,471,806 | 649,000,191 | 163,202,201 | 94,488,489 | 0 | 814,989,422 | 1,721,680,303 | 27,328,414 |
| optimized-2-v2 | 560,637,052 | 857,471,806 | 649,000,191 | 163,202,201 | 94,488,489 | 0 | 814,989,422 | 1,721,680,303 | 27,328,414 |
| optimized-2-v3 | 470,994,406 | 694,339,163 | 534,629,122 | 123,891,439 | 84,076,059 | 0 | 614,427,236 | 1,357,023,856 | 25,006,719 |

*Πίνακας 8 - Εντολές και Κύκλοι του επεξεργαστή μαζί με την μνήμη.*

Παρατηρούμε ότι το μέγεθος της ROM μνήμης μειώνεται όλο και πιο πολύ , οπότε δοκιμάζοντας μειώσαμε και την μνήμη που δηλώνουμε στα memory.map και scatter.txt . Επίσης δηλώνοντας προσεκτικά τις βάσεις των stack , heap προκύπτει ο παρακάτω συγκεντρωτικός πίνακας:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ROM | | | RAM | | | | | Total MB |
|  | Start Address | End Address | MB | Start Address | Heap Base | Stack Base | End Address | MB |  |
| initial | 0x00 | 0x04000000 | 64 | 0x04000000 | 0x06000000 | 0x08000000 | 0x10000000 | 192 | 256 |
| optimized-1 | 0x00 | 0x01000000 | 16 | 0x01000000 | 0x01000000 | 0x02000000 | 0x04000000 | 48 | 64 |
| optimized-2-v1 | 0x00 | 0x01000000 | 16 | 0x01000000 | 0x01000000 | 0x02000000 | 0x04000000 | 48 | 64 |
| optimized-2-v2 | 0x00 | 0x01000000 | 16 | 0x01000000 | 0x01000000 | 0x02000000 | 0x04000000 | 48 | 64 |
| optimized-2-v3 | 0x00 | 0x00800000 | 8 | 0x00800000 | 0x00800000 | 0x01000000 | 0x02000000 | 24 | 32 |

*Πίνακας 9 - Δομή μνήμης των προγραμμάτων με μνήμη*

Έχουμε φτάσει την συμπιεσμένη μνήμη ROM στα 8 MB και την μνήμη RAM κατά την εκτέλεση του προγράμματος (και μετά την αποσυμπίεση) στα 24 MB. Έχουμε κάνει σχεδόν ότι ήταν δυνατό για να μειώσουμε το μέγεθος της απαιτούμενης μνήμης. Τώρα ας κοιτάξουμε την δομή και την ταχύτητα της μνήμης.

Αρχικά θα δοκιμάσουμε να μειώσουμε την ταχύτητα που χρειάζεται ο επεξεργαστής για την επικοινωνία με την μνήμη. Έχουμε λοιπόν τον παρακάτω πίνακα με διαφορετικά αρχεία .map για την έκδοση v1 του βελτιστοποιημένου κώδικα:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Base | Dec | From .map file : | | | |  |  |  |  |  |  |  |
| cw-optimized-2-v1 | name | Nread\_ns | Nwrite\_ns | Sread\_ns | Swrite\_ns | ns | s | Wait\_States | Total | clock | cputime | sys\_clock |
| memory.map | ROM | 1 | 1 | 1 | 1 | 87764820 | 14 | 814989422 | 1721680303 | 34433606 | 1721680303 | 3443 |
| RAM | 250 | 250 | 50 | 50 | 456071460 | 18 |
| memory2.map | ROM | 1 | 1 | 1 | 1 | 87764820 | 14 | 1629978844 | 2536669725 | 50733394 | 2536669725 | 5073 |
| RAM | 500 | 500 | 100 | 100 | 755859900 | 34 |
| memory3.map | ROM | 1 | 1 | 1 | 1 | 87764820 | 14 | 407494711 | 1314185592 | 26283711 | 1314185592 | 2628 |
| RAM | 125 | 125 | 25 | 25 | 306177240 | 10 |
| memory4.map | ROM | 1 | 1 | 1 | 1 | 87764820 | 14 | 910745500 | 1817436381 | 36348727 | 1817436381 | 3634 |
| RAM | 250 | 250 | 100 | 100 | 371193020 | 20 |
| memory5.map | ROM | 1 | 1 | 1 | 1 | 659101960 | 11 | 767111383 | 1673802264 | 33476045 | 1673802264 | 3347 |
| RAM | 250 | 250 | 25 | 25 | 498510680 | 17 |

*Πίνακας 10 - Μετρήσεις της μνήμης του βελτιστοποιημένου κώδικα v1*

Αντίστοιχες μετρήσεις πήραμε για την έκδοση v2 και την v3:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Base | Dec | From .map file : | | | |  |  |  |  |  |  |  |
| cw-optimized-2-v2 | name | Nread\_ns | Nwrite\_ns | Sread\_ns | Swrite\_ns | ns | s | Wait\_States | Total | clock | cputime | sys\_clock |
| memory.map | ROM | 1 | 1 | 1 | 1 | 87764820 | 14 | 814989422 | 1721680303 | 34433606 | 1721680303 | 3443 |
| RAM | 250 | 250 | 50 | 50 | 456071460 | 18 |
| memory2.map | ROM | 1 | 1 | 1 | 1 | 87764820 | 14 | 1629978844 | 2536669725 | 50733394 | 2536669725 | 5073 |
| RAM | 500 | 500 | 100 | 100 | 755859900 | 34 |
| memory3.map | ROM | 1 | 1 | 1 | 1 | 87764820 | 14 | 407494711 | 1314185592 | 26283711 | 1314185592 | 2628 |
| RAM | 125 | 125 | 25 | 25 | 306177240 | 10 |
| memory4.map | ROM | 1 | 1 | 1 | 1 | 87764820 | 14 | 910745500 | 1817436381 | 36348727 | 1817436381 | 3634 |
| RAM | 250 | 250 | 100 | 100 | 371193020 | 20 |
| memory5.map | ROM | 1 | 1 | 1 | 1 | 87764820 | 14 | 767111383 | 1673802264 | 33476045 | 1673802264 | 3347 |
| RAM | 250 | 250 | 25 | 25 | 498510680 | 17 |

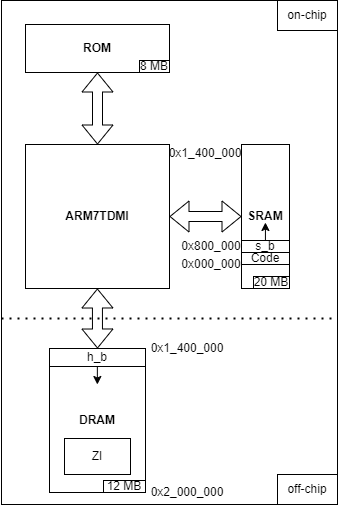
*Πίνακας 11 - Μετρήσεις της μνήμης του βελτιστοποιημένου κώδικα v2*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Base | Dec | From .map file : | | | |  |  |  |  |  |  |  |
| cw-optimized-2-v3 | name | Nread\_ns | Nwrite\_ns | Sread\_ns | Swrite\_ns | ns | s | Wait\_States | Total | clock | cputime | sys\_clock |
| memory.map | ROM | 1 | 1 | 1 | 1 | 659101960 | 11 | 614427236 | 1357023856 | 27140477 | 1357023856 | 2714 |
| RAM | 250 | 250 | 50 | 50 | 799853980 | 13 |
| memory2.map | ROM | 1 | 1 | 1 | 1 | 659101960 | 11 | 1228854472 | 1971451092 | 39429021 | 1971451092 | 3942 |
| RAM | 500 | 500 | 100 | 100 | 88398700 | 26 |
| memory3.map | ROM | 1 | 1 | 1 | 1 | 659101960 | 11 | 307213618 | 1049810238 | 20996204 | 1049810238 | 2099 |
| RAM | 125 | 125 | 25 | 25 | 655581620 | 7 |
| memory4.map | ROM | 1 | 1 | 1 | 1 | 659101960 | 11 | 67289800 | 1415495520 | 28309910 | 1415495520 | 2830 |
| RAM | 250 | 250 | 100 | 100 | 969287260 | 14 |
| memory5.map | ROM | 1 | 1 | 1 | 1 | 659101960 | 11 | 585191404 | 1327788024 | 26555760 | 1327788024 | 2655 |
| RAM | 250 | 250 | 25 | 25 | 215137340 | 13 |

*Πίνακας 12 – Μετρήσεις της μνήμης του βελτιστοποιημένου κώδικα v3*

Παρατηρούμε ότι η καλύτεροι χρόνοι είναι όταν η διεπαφή της μνήμης είναι στα 125 ns και 25 ns για τις μη-διαδοχικές και τις διαδοχικές εγγραφές/αναγνώσεις αντίστοιχα.

Θεωρώντας ότι είναι μία μνήμη εκτός του chip δεν μπορύμε παρά να “διορθώσουμε” αυτή την θεώρηση. Έτσι λοιπόν παίρνουμε τους παραπάνω χρόνους και τους αναθέτουμε στην καινύργια DRAM μας (off-chip) και θεωρούμε και μία on-chip μνήμη την SRAM. Έτσι πλέον μπορούμε να έχουμε την παρακάτω δομή:



*Εικόνα 5 - Δομή της Μνήμης με on-chip και off-chip μνήμες*

Να σημειωθεί ότι το διπλό βέλος μεταξύ του επεξεργαστή και της μνήμης ROM υποδεικνύει τις δύο λειτουργίες: την αποσυμπίεση της μνήμης LOAD\_ROM και την ανάγνωση των δεδομένων μόνο για ανάγνωση.

Μια επιπλέον διαφοροποίηση με την προηγούμενη δομή της μνήμης είναι ότι τοποθετήσαμε τα Zero-Initialized δεδομένα μέσα στην DRAM και μέσα στην SRAM κρατήσαμε την ίδια δομή με πριν της stack και της heap.

Παρατηρούμε από τον Πίνακα-9 ότι τα wait states (καταστάσεις αναμονής του διάυλου και του επεξεργαστή) είναι 1,240,908,894 και φτάνουν στην έκδοση 3 του κώδικα τις 614,427,236. Αν εκμεταλλευτούμε την νέα δομή της μνήμης, έχουμε:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| WITH MEMORY | Instuctions | Core Cycles | S Cycles | N Cycles | I Cycles | C Cycles | Wait Cycles | Total | True Idle Cycles |
| scatter.txt | 470,994,406 | 694,339,163 | 534,629,122 | 123,891,439 | 84,076,059 | 0 | 614,427,236 | 1,357,023,856 | 25,006,719 |
| scatter2.txt | 470,994,406 | 694,339,163 | 534,629,122 | 123,891,439 | 84,076,059 | 0 | 0 | 742,596,504 | 25,006,716 |
| scatter3.txt | 470,994,406 | 694,339,163 | 534,629,122 | 123,891,439 | 84,076,059 | 0 | 0 | 742,596,504 | 25,006,716 |

*Πίνακας 13 - Εντολές και Κύκλοι του Επεξεργαστή μαζί με τις 3 διαφορετικές δομές μνήμης*

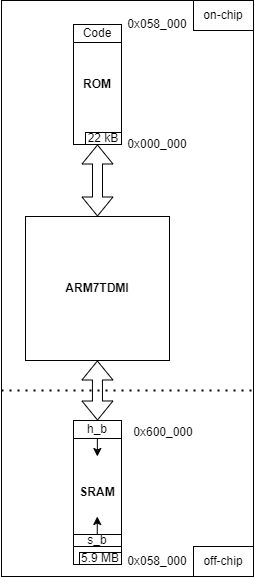
Οι μετρήσεις της μνήμης φαίνονται αναλυτικά παρακάτω:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Base | Hex | From scatter.txt : | | From .map file : | | | |  |  |  |  |  |  |
| cw-optimized-2-v3 | name | start | limit | Nread\_ns | Nwrite\_ns | Sread\_ns | Swrite\_ns | Nreads | Nwrites | Sreads | Swrites | ns | s |
| scatter.txt | ROM | 0 | 800000 | 1 | 1 | 1 | 1 | 49F7FD0 | 0 | 1E1FB08A | 0 | 53B2F54 | E |
| RAM | 800000 | 2000000 | 7D | 7D | 19 | 19 | 1A3DE69 | 11F10B6 | C518D5 | F901A3 | 123FE4D8 | A |
| scatter2.txt | ROM | 0 | 8000000 | 1 | 1 | 1 | 1 | 49F7FBA | 0 | 1E1FB056 | 0 | 2749340 | B |
| SRAM | 800000 | 1400000 | 1 | 1 | 1 | 1 | 1A3DE5D | 11F10AD | C518D5 | F9019E | 1E79F360 | 1 |
| DRAM | 1400000 | 2000000 | 7D | 7D | 19 | 19 | 0 | 0 | 0 | 0 | 0 | 0 |
| scatter3.txt | ROM | 0 | 800000 | 1 | 1 | 1 | 1 | 1F4D41D | 19DB953 | 13A7488 | 1A01B0F | 1B2F1924 | 12 |
| SRAM | 800000 | 1400000 | A | A | A | A | 1A3DE5D | 11F10AD | C518D5 | F9019E | 1E79F360 | 1 |
| DRAM | 1400000 | 2000000 | 7D | 7D | 19 | 19 | 0 | 0 | 0 | 0 | 0 | 0 |

*Πίνακας 14 - Μετρήσεις της μνήμης του βελτιστοποιημένου κώδικα v3 με τις διαφορετικές δομές μνήμης*

Παρατηρούμε ότι οι εγγραφές και ανγνώσεις στην μνήμη DRAM είναι μηδενικές. Όποτε είναι επιπλέον χώρος που δεν χρησιμοποιείται.

Επαναδιατάζουμε την μνήμη:



*Εικόνα 6 - Δομή της Μνήμης on-chip ROM και off-chip SRAM μνήμες*

Αυτή την φορά δεν σπάμε την μνήμη σε δύο κομμάτια. Και έχουμε τα παρακάτω αποτελέσματα συγκριτικά με τις προηγούμενες δομές:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| WITH MEMORY | Instuctions | Core Cycles | S Cycles | N Cycles | I Cycles | C Cycles | Wait Cycles | Total | True Idle Cycles |
| scatter.txt | 470,994,406 | 694,339,163 | 534,629,122 | 123,891,439 | 84,076,059 | 0 | 614,427,236 | 1,357,023,856 | 25,006,719 |
| scatter2.txt | 470,994,406 | 694,339,163 | 534,629,122 | 123,891,439 | 84,076,059 | 0 | 0 | 742,596,504 | 25,006,716 |
| scatter3.txt | 470,994,406 | 694,339,163 | 534,629,122 | 123,891,439 | 84,076,059 | 0 | 0 | 742,596,504 | 25,006,716 |
| scatter4.txt | 470,994,406 | 694,339,163 | 534,629,122 | 123,891,439 | 84,076,059 | 0 | 243,790,188 | 986,386,808 | 25,006,719 |

*Πίνακας 15 - Εντολές και Κύκλοι των 4ων διαφορετικών δομών μνήμης*

Και για την μνήμη παίρνουμε τις παρακάτω μετρήσεις:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Base | Hex | From scatter.txt : | | From .map file : | | | |  |  |  |  |  |  |
| cw-optimized-2-v3 | name | start | limit | Nread\_ns | Nwrite\_ns | Sread\_ns | Swrite\_ns | Nreads | Nwrites | Sreads | Swrites | ns | s |
| scatter4.txt | ROM | 0 | 58000 | 1 | 1 | 1 | 1 | 49F7FD0 | 0 | 1E1FB08A | 0 | 27491908 | B |
| SRAM | 58000 | 600000 | 64 | 64 | 32 | 32 | 1A3DE69 | 11F10B6 | C518D5 | F901A3 | 1712E03C | 6 |

*Πίνακας 16 - Μετρήσεις της μνήμης του βελτιστοποιημένου κώδικα v3 μαζί με την βέλτιστη δομή μνήμης*

Για τους χρόνους που έτρεξε ο επεξεργαστής για τα παραπάνω scatter αρχεία πήραμε τα παρακάτω αποτελέσματα:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Base | Dec |  |  |  |
| cw-optimized-2-v3 | name | clock (s) | cputime (in clock units) | sys\_clock |
| scatter4.txt | ROM | 19.727736 | 986386808 | 1972 |
| SRAM |
| scatter3.txt | ROM | 14.85193 | 742596504 | 1485 |
| SRAM |
| DRAM |
| scatter2.txt | ROM | 14.85193 | 742596504 | 1485 |
| SRAM |
| DRAM |
| scatter.txt | ROM | 26.283711 | 1314185592 | 2628 |
| RAM |

*Πίνακας 17 - Χρόνοι Εκτέλεσης*

Θα μπορούσαμε να διαλέξουμε μία SRAM ή μία DRAM που να έχει μέγεθος 16Mbits και να μπορούμε να διαβάσουμε και να γράψουμε με χρόνους 100 ns και 50 ns για μη διαδοχικούς και διαδοχικούς κύκλους αντίστοιχα.

# Αναφορές

1. Engineering A Compiler by Keith D. Cooper and Linda Torczon, 3rd Edition, Elsevier publications, 2023
2. Digital Image Processing by R. Gonzalez and R. Woods, 4th Edition, Pearson, 2018
3. Digital Video and HD - Algorithms and Interfaces by Charles Poynton, 2nd Edition, Elsevier publications, 2012
4. Compilers: Principles, Techniques and Tools by Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman, 2nd Edition, Addison-Wesley, 2006
5. Best Practice Guide - Deep Learning - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Schematic-illustration-of-a-convolutional-operation-The-convolutional-kernel-shifts-over\_fig2\_332190148 [accessed 26 Nov, 2023]
6. Computer Organization and Architecture – Design for Performance by William Stallings, 11th Edition, Pearson, 2019
7. Embedded System Design – Embedded Systems Foundations of Cyber-Physical Systems and the Internet of Things by Peter Marwedel, 4th Edition, Springer, 2021
8. ARM System Developer’s Guide Designing and Optimizing System Software by Andrew N. Sloss, Dominic Symes, Chris Wright, Elsevier publications, 2004